Improving RL Power for On-Line Evolution of Gaits in Modular Robots

Jelisavcic, Milan; De Carlo, Matteo; Haasdijk, Evert; Eiben, A.E.

published in
2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016
2016

DOI (link to publisher)
10.1109/SSCI.2016.7850166

document version
Publisher's PDF, also known as Version of record

Improving RL Power for On-Line Evolution of Gaits in Modular Robots

Milan Jelisavcic, Matteo De Carlo, Evert Haasdijk, and A.E. Eiben
Computational Intelligence Group
Department of Computer Science
Vrije Universiteit Amsterdam
Email: milan.jelisavcic@gmail.com

***Abstract*—This paper addresses the problem of on-line gait learning in modular robots whose shape is not known in advance. The best algorithm for this problem known to us is a reinforcement learning method, called RL PoWER. In this study we revisit the original RL PoWER algorithm and observe that in essence it is a specific evolutionary algorithm. Based on this insight we propose two modifications of the main search operators and compare the quality of the evolved gaits when either or both of these modified operators are employed. The results show that using 2-parent crossover as well as mutation with self-adaptive step-sizes can significantly improve the performance of the original algorithm.**

I. Introduction

The work reported in this paper is part of a larger research programme towards the Evolution of Things as outlined in [1]–[3]. In particular, we are developing and investigating robotic systems where robot morphologies and controllers can evolve in real-time and real-space. A theoretical model of such systems, called the Triangle of Life, has been introduced in [4] distinguishing three principal stages: Morphogenesis, Infancy, and Mature Life as illustrated in Figure 1.

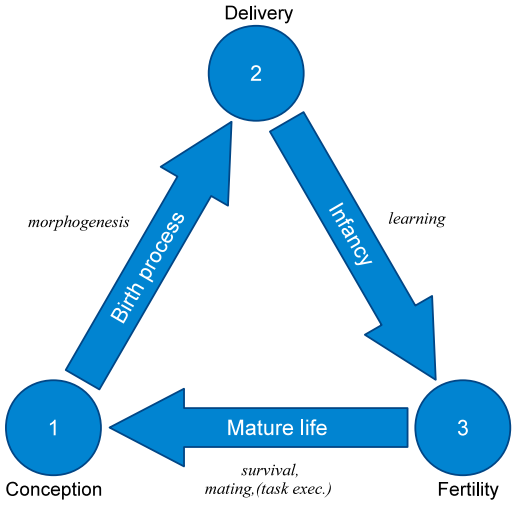

Fig. 1. The Triangle of Life. The pivotal moments that span the triangle and separate the three stages are: 1) Conception: A new genome is activated, construction of a new robot starts. 2) Delivery: Construction of the new robot is completed. 3) Fertility: The robot becomes an adult, ready to conceive offspring.

There are many possible implementations of the general Triangle of Life (ToL) framework, distinguishable by different morphologies and controller architectures, but in all of these newborn robots are random combinations of the bodies and minds of their parents. This raises a problem: new robots are born with new bodies that can and will be different from the bodies of the parents. This implies that every newborn robot must acquire a new controller that matches the new body quickly after birth.[1]

In this paper we focus on a specific case of this problem, that of gait learning in modular robots whose shape is not known in advance. The general question we address is: How can a modular robot learn a good gait quickly? Technically speaking, we are interested in a general gait learning algorithm that works for any given robot within the space of all possible morphologies constructible with the modules we use. Our algorithm of choice is the RL PoWER algorithm as proposed by Kober and Peters [6] and employed for gait learning in Roombots in our previous work [7].

In particular, we pursue two research objectives:

- To see whether RL PoWER also works for modular robots based on RoboGen [8], instead of Roombots.
- To increase the performance of RL PoWER by altering the main search operators used therein.

The basis for the second objective is (re)describing RL PoWER as an evolutionary algorithm (EA) with a specific mutation and crossover operator. Considering RL PoWER from an EA perspective provides hints for possible improvements by using a different crossover, a different mutation, or both. Therefore, we implement three new versions of RL PoWER and assess the performance of these on a test suite of 12 robots in simulation. The results confirm that RL PoWER works in this new type of robots and we observe that the new search operators improve its performance.

II. Related Work

The design of a good and fast gait-learning method is not a trivial task. Locomotion requires the creation of rhythmic patterns which satisfy multiple constraints: generating forward

[1]Even if the parents had well matching bodies and minds, there is no general guarantee that recombination and mutation will keep the good match. See also [5]

motion, without falling over, with low energy, possibly coping with different environments, hardware failures, changes in the environment and/or of the organism, as Spröwitz stated [9].

An early approach is based on gait control tables where a control table is an array of control sequences for an actuators with transitional conditions between sequences [10], [11]. Another approach exploits central pattern generators (CPG), which models circuitry found in vertebrates that outputs cyclic patterns [12]. The actuators of a robot are controlled by the signal generated by coupled synchronised CPGs which allows them to synchronise their movement. Another popular approach is based on neural networks (NN), and especially on HyperNEAT [13]–[15]. Looking at the performance of investigated techniques has shown that they produce well performing and stable gaits on both non-modular and modular organisms [16]–[18]. In summary, HyperNEAT can produce efficient gaits, but the costs of learning time are too high for a real-time application.

The RL PoWER is reinforcement learning algorithm described by Kober and Peters [**?**], [6]. The properties of this algorithm were investigated in comparison to HyperNEAT and Simulated Annealing [7], [19]. These studies have shown that RL PoWER is a superior method for on-line gait learning since it converges quickly to learn sufficiently good gaits in a short time. In this article we investigate different variations of the original algorithm and present the details in subsequent sections.

III. Robot Description

The robots we use here are based on the RoboGen system specifically designed for evolutionary robotics studies [8]. The main advantage of this system is that any given robot can be easily constructed in the real world through assembling 3D-printed and prefabricated modules (e.g., servos, logic boards, batteries). We altered the original RoboGen design by making the 'head' module bigger and using Raspberry Pi 2 boards, instead of Arduinos for increased computing power.

Further to changing the morphologies, we also changed the controllers and employ a set of cyclic splines that define an open-loop gait. Each controller, called policy in the sequel, consists of one spline for each active joint (servo). Thus, in total we have as many splines as there are servos, where each spline specifies the angular positions of a servo over a certain amount of time. Formally, a cyclic spline is a mathematical function that is defined using a set of n control points. Each control point is defined by (t_i, α_i) where t_i represents time and α_i the corresponding value. $t_i \in [0, 1]$ is defined as

$$t_i = \frac{i}{n-1}, \forall i = 0, \ldots, (n-1) \tag{1}$$

and $\alpha_i \in [0, 1]$ is freely defined. An additional control point (t_n, α_n) is defined to enforce that the last value is equal to the first, i.e. $\alpha_0 = \alpha_n$ and so enable cyclic splines. These control points are then used to interpolate a cubic spline with periodic boundary conditions using GSL [20] dedicated C functions. Using GSL it is possible to query a spline for a different number of points than it was defined with. This use of sets of cyclic spline functions as the representation was taken from [21]. The task of the learning algorithm is then to optimise the parameters of a set of splines so that performance –distance travelled by the given robot in our case– is maximised.

IV. Gait Learning Algorithm

RL PoWER has been introduced for optimising parameters of cyclic splines based on an Expectation-Maximization approach. The algorithm creates the initial policy with one spline per servo, each having 2 control points. These control points are initialised at 0.5 and then perturbed using Gaussian noise $N(0, \sigma)$. The algorithm then enters an evaluation–adaptation loop to refine the policy, until the stopping condition is reached.

Evaluation of a policy is carried out by using it to control the robot and measuring the distance travelled over a given period of time. The reward awarded to a policy is calculated as:

$$R = \left(100 \frac{\sqrt{\Delta_x^2 + \Delta_y^2}}{\Delta_t} \right)^6 \tag{2}$$

where Δ_x and Δ_y is the displacement over the x and y axes measured in meters and Δ_t the evaluation time.

Adaptation consists of three components: spline size increase, exploitation and exploration. The spline is gradually refined by incrementing the number of control points periodically as proposed in [21] In the exploitation step, the current parameters are adapted based on the values of the k best policies encountered so far. These k best policies $P_1, \ldots, P_k$ are kept in a ranked list that is updated after each evaluation and they are used to create a new policy P by

$$P = w_1 \cdot P_1 + \cdots + w_k \cdot P_k \tag{3}$$

The used weights are reward proportional, defined as

$$w_i = \frac{R_i}{\epsilon + \sum_{j=1}^{k} R_j}, \tag{4}$$

where R_i is the reward of P_i and ϵ is a parameter to avoid division by 0, set to 10^{-10}.

In the exploration phase policies are adapted by applying Gaussian perturbation to every control point in the policy resulting from exploitation. Over the course of the run the variance σ^2 is diminished which decreases exploration and increases exploitation.

The operating parameters for RL PoWER, such as the variance and its decay factor, as well as the reward function, were taken from [21]. The values were: 0.008 for the variance and 0.98 for the variance decay, k is set to 10 as it was used in previous research [7]. The splines are initialised with 2 control points and are allowed to grow to a maximum of 100 control points over the course of a run, in this case the spline is grown every 10 evaluations ($round(\frac{1000}{100-2} = 10)$. ϵ is a parameter to avoid division by 0.

Algorithm 1: RL PoWER

```
initialisation;
policy ← initialisation;
evaluate(policy);
while evaluation < total_evaluations do
    /* Update the ranking of k best
       policies                                  */
    ranking.insert(policy);
    if ranking.size > k then
        ranking.remove_worst();
    end
    /* Spline size increase                      */
    if evaluation mod increase_delta = 0 then
        spline_size ← spline_size + 1;
        reinterpolate_all(ranking);
        reinterpolate(policy);
    end
    /* Exploitation                              */
    rewards ← 0;
    weighted_total ← 0;
    for p in ranking do
        rewards ← rewards + p.reward;
        weighted_parameters ← p.reward *
          (policy.parameters - p.parameters);
        weighted_total ← weighted_total +
          weighted_parameters;
    end
    next_policy.parameters ← policy.parameters +
      weighted_total / (rewards + ϵ);
    /* Exploration                               */
    next_policy.parameters ← next_policy.parameters +
      normrnd(0,sqrt(variance));
    policy ← next_policy;
    variance ← variance * variance_decay;
    evaluate(policy);
end
```

RL PoWER can be viewed as evolutionary algorithm with policies as individuals, fitness defined as the corresponding reward, population size k, an elitist $(k+1)$ selection strategy, a k-parent crossover, and Gaussian mutation [22]. This evolutionary perspective provides hints for possible improvements of the algorithm by altering the crossover and mutation operators. Multi-parent crossovers are known to lead to fast convergence to local optima [23], suggesting that using two parents instead of ten may increase overall performance. Obviously, if not all ten population members are involved in crossover, then we need a parent selection mechanism as well. To this end, we use binary tournament selection that is a common and often good choice for selecting parents [24]. Thus, to select the two parents we execute two independent 2-tournaments. In each tournament two individuals are chosen with uniform distribution from the population of 10 and the best of these two becomes a parent.

Regarding mutation, using self-adaptive step-sizes (σs) is a promising option that is known to work well for numerical optimization problems. The simplest version of this method uses one step size globaly. This σ is mutated each time step *before* using it to create a new individual by multiplying it by a term $\exp^{\Gamma}$ where Γ is random variable from normal distribution;

$$\sigma' = \sigma \cdot \exp^{\tau \cdot N(0,1)} \tag{5}$$

The constant τ is external parameter that is set by user and we set it to 0.2 in our experiments to have fine-grained adaptation.

Based on these modifications we obtain four gait learning algorithms:

- Algorithm A: the original RL PoWER
- Algorithm B: RL PoWER with 2-parent crossover
- Algorithm C: RL PoWER with self-adaptive σ
- Algorithm D: the combination of B and C.

V. Experiments

To evaluate the gain learning algorithms quickly, all experiments were done in simulation. To this end we used Revolve, the **R**obot **Evolve** toolkit developed at our department. Revolve is a set of Python and C++ libraries created to aid in setting up simulation experiments involving robots with evolvable bodies and/or minds. It builds on top of the Gazebo simulator, complementing this simulator with a set of tools that aim to provide a convenient way to set up such experiments. Revolve's source code with RL PoWER implementation can be found at *https://www.github.com/milanjelisavcic/revolve*.

Table 2 shows the 12 robot shapes used for our experiments divided by shape and size. Three of the shapes are named after animals they resemble, spider, gecko, and snake, and each comes in three different sizes. This yields 9 different robots, all symmetrical. They have interlocked joints by $90°$ which gives them one more degree of freedom on every increase in size. Because of the nature of RL PoWER algorithm where an independent spline is generated for each joint there is no direct interactions or constraints among them. All joint behaviors evolve independently and only the final outcome of their combined movement in form of a speed the body induces indirectly influences evolution of a next policy. Distinguishing robots by size enables us to investigate whether increasing the robot size, hence the freedom, increases the speed of the evolved gaits.

In addition, we have three asymmetrical shapes that are obtained by making a crossover between the symmetrical shapes. The rationale for this lies in the general motivational scenario explained in the Introduction, a population of self-reproducing robots, where crossover between different morphologies is possible. $BabyA$ is formed from $gecko7$ and $spider9$, $babyB$ is formed from $snake7$ and $spider13$, and $babyC$ was formed from $snake9$ and $gecko17$. These forms were also introduced to test how does this algorithm behave on asymmetrical morphologies and to check weather asymmetry could be seen as a defect considering gait learning problem.

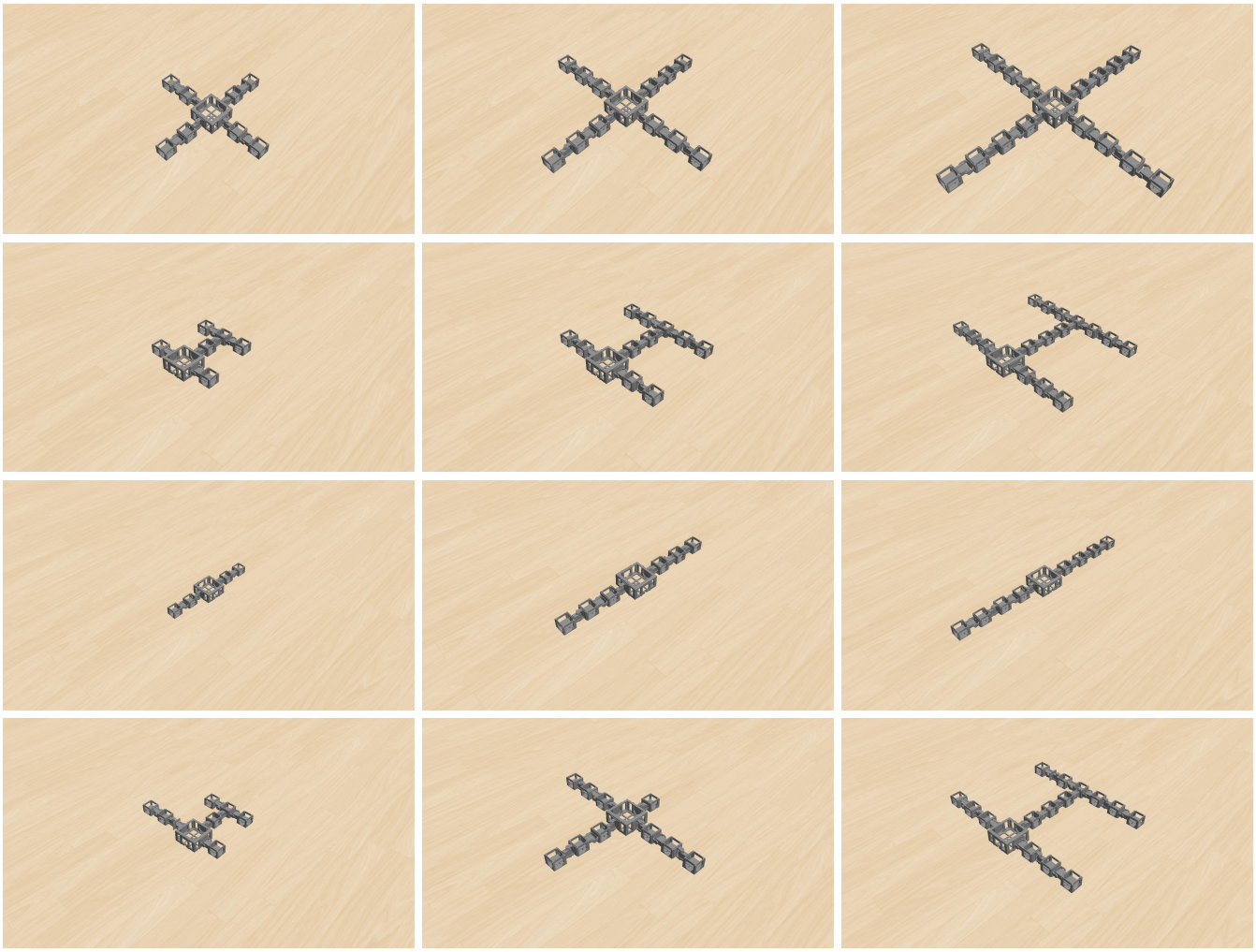

Fig. 2. Images of the used morphologies. The top three rows exhibit the basic shapes named spider, gecko, and snake in three sizes. The bottom row shows the three 'baby' morphologies created through recombining basic shapes.

The implementation of RL PoWER is a C++ plugin that is loaded in the Gazebo environment. Once loaded, the plugin controls the behaviour of a given robot. It is important to note that controller evolution is done in on-line manner, which means that every new evaluation starts from a position of joints where previous evaluation finished. This enables us to test our methods in real hardware, where resetting everything to a starting position for each evaluation is not practicable.

We run each gait learning algorithm on each shape ten times independently with a new random seed. In every run we record the best fitness values after every evaluation up to 1000 evaluations which is a stop condition for every experiment. Best fitnesses after every evaluation were averaged throughout ten trials and results are presented in Section VI.

VI. Results

The performance of the algorithms is exhibited in Figure 3 for the spider-like, gecko-like, snake-like and three asymmetrical shapes respectively. Each column in the figure contains four plots for every version of the tested algorithm. Each plot contains three curves that shows the performance for same shape in different sizes. Comparing plots for Algorithm A and B shows difference in performance; with a slightly faster convergence in Algorithm A. This shows that adding modifications to the crossover operator indeed decreases the convergence speed, but also leads to a better solution. Comparing plots for Algorithm A and C shows improvements in reaching better solution, but in a longer time period. Comparing plots for Algorithm A, C and D shows that combining the modified crossover operator to the self-adaptive sigma does provide an added value. Results indicate that modified crossover operator does improve performance, as well as adding self-adaptive sigma, but the results improve the best using the combination of these modifications.

For a closer analysis, Table I shows the mean best fitness after 1000 evaluations for each algorithm. The best performance for each morphology are highlighted in grey. Algorithms C and D (the variants with self-adaptive σ) yield the best performance, but the variance of the performances is substantial. Table II highlights the effects of the modifications

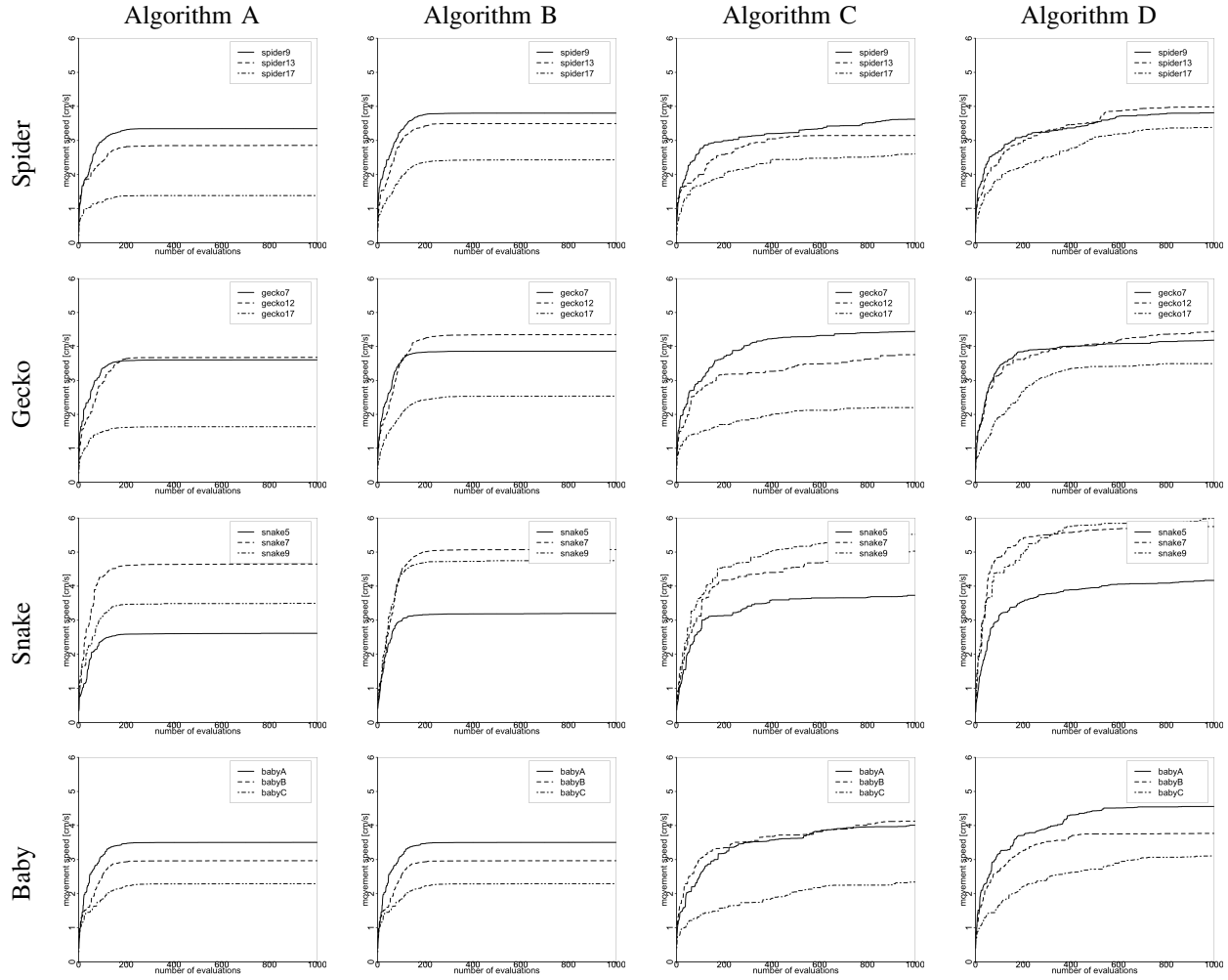

Fig. 3. Mean best fitnesses over time. Each plot shows the mean best fitness for three sizes of a shape (one shape per row, as indicated in the first column) one of the algorithms (one algorithm per column, as indicated in the first row).

in algorithms B,C, and D as the change in performance vis a vis unmodified RL PoWER (algorithm A).

TABLE II
IMPROVEMENT OF MEAN BEST FITNESS VALUES IN PERCENTAGES COMPARED TO THE STANDARD RL POWER IMPLEMENTATION (ALG. A), BASED ON THE DATA IN TABLE I

	Alg. B	Alg. C	Alg. D
spider 9	13	8	14
spider 13	22	10	40
spider 17	76	89	1.46
gecko 7	7	23	16
gecko 12	18	2	21
gecko 17	55	34	1.13
snake 5	22	42	59
snake 7	9	8	24
snake 9	36	58	71
babyA	21	14	30
babyB	26	39	27
babyC	0	2	36
Avg.	25.41	27.41	49.75

TABLE III
AVERAGE NUMBER OF EVALUATIONS WHEN MAXIMUM SPEED IS REACHED.

	Alg. A	Alg. B	Alg. C	Alg. D
spider 9	288.7	446.2	594.8	659.3
spider 13	370.6	333.6	381.8	523.5
spider 17	145.2	424.3	606.7	686.8
gecko 7	520.6	326.6	643.9	549.6
gecko 12	431.2	529.7	589.4	608.4
gecko 17	163.4	425.6	484.7	496.6
snake 5	404.3	495.6	414.7	691.5
snake 7	365.2	426.3	581.9	484.0
snake 9	262.9	361.6	686.6	618.0
babyA	285.9	592.3	710.3	594.7
babyB	278.3	560.7	576.6	494.4
babyC	228.5	402.4	665.1	610.0
Avg.	312.06	443.74	578.04	584.73

To determine whether the differences in performance between the algorithms are statistically significant, we used the Kruskal-Wallis test ($p = 3.51 \times 10^{-10}$), followed by Dunn's test to determine which algorithms differ. The results of these

TABLE I

MEAN BEST FITNESS VALUES (IN CM S^{-1}) AND STANDARD DEVIATIONS FOR THE FOUR ALGORITHMS. RESULTS ARE AVERAGED OVER 10 REPLICATE RUNS PER ROBOT. 'SPIDER 9' INDICATES THE SPIDER-SHAPED ROBOT OF 9 COMPONENTS, 'SPIDER 13' THE SAME OF 13 COMPONENTS, AND SO ON, AS SHOWN IN FIGURE 3. THE BEST RESULT FOR EACH SHAPE IS HIGHLIGHTED IN GREY.

	Algorithm A		Algorithm B		Algorithm C		Algorithm D	
	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.
spider 9	3.35	0.72	3.80	0.41	3.62	0.85	3.81	0.69
spider 13	2.86	0.59	3.50	0.54	3.15	0.84	3.99	0.59
spider 17	1.38	0.64	2.43	0.65	2.61	0.96	3.39	0.70
gecko 7	3.61	0.76	3.86	0.74	4.44	0.26	4.18	0.61
gecko 12	3.68	0.95	4.35	1.04	3.76	1.04	4.44	0.87
gecko 17	1.64	0.61	2.54	0.33	2.20	0.66	3.49	1.33
snake 5	2.62	1.23	3.20	1.08	3.73	0.95	4.17	0.90
snake 7	4.65	1.57	5.07	1.20	5.03	1.61	5.75	0.97
snake 9	3.50	1.90	4.75	1.96	5.52	1.88	6.00	1.06
babyA	3.51	1.07	4.24	0.49	4.01	0.73	4.56	0.40
babyB	2.97	1.14	3.75	0.84	4.13	0.67	3.77	1.01
babyC	2.29	0.91	2.29	0.72	2.34	1.13	3.11	0.57

tests are summarised in Table IV.

TABLE IV

P VALUES FROM DUNN'S TEST COMPARING PERFORMANCE FROM ALL RUNS OF EACH ALGORITHM. ALGORITHMS C AND D ARE SIGNIFICANTLY BETTER THAN A AND B.

	Alg. A	Alg. B	Alg. C	Alg. D
Alg. A		2.01×10^{-3}	3.56×10^{-4}	4.82×10^{-11}
Alg. B			1.0	2.53×10^{-3}
Alg. C				1.15×10^{-2}
Alg. D				

The first row of Table IV shows that the new algorithm variants B, C, and D significantly outperform the original one. Comparing B and C we see no significant difference. That is, no difference considering the end results. However, the plots in Figure 3 show that B learns quicker. Looking at the last column we can observe that D significantly differs from A, B, and C. In particular, the superior performance displayed in Table I is not a random effect. This means that the combination of the two-parent crossover and the self-adaptive mutation step-sizes yield a significant boost in performance.

To gain additional insights in the behaviour of the system we investigated the development of genetic diversity over time. To this end, we divided the total duration of a run into epochs. An epoch is a period where the number of control points is not changing. Since RL PoWER adds one control point after every 100 evaluations (per spline), we have 10 epochs $e_1, \ldots, e_{10}$, where e_1 runs from evaluation 1 to 99, e_2 runs from evaluation 100 to 199, etc., until e_{10} running from evaluation 900 through 999.

We define the genetic diversity *for a given control point* as the standard deviation of all values for that control point in the given population of ten policies. Formally, for each point in time t, we have n_t control points $c_1, \ldots, c_{n_t}$. The value of c_i in a given policy p_j ($j = 1, \ldots, 10$) is then $cp_{i,j}(t)$. Thus, the standard deviation for c_i is $\sigma_{c_i}(t)$:

$$\sigma_{c_i}(t) = \sqrt{\frac{\sum_{j=1}^{10} (cp_{i,j}(t) - \mu_i(t))^2}{10}} \tag{6}$$

where $\mu_i(t)$ is mean value of $cp_{i,j}(t)$ for j.

$$\mu_i(t) = \frac{1}{10} \cdot \sum_{j=1}^{10} cp_{i,j}(t) \tag{7}$$

For a moment t we then calculate averaged value M_t for σ_c:

$$M_t = \frac{1}{n_t} \cdot \sum_{c=1}^{n_t} \sigma_{c_i} \tag{8}$$

Figure 4 exhibits the mean value M further averaged over all experimental runs. The graphs show that using self-adaptive mutation step-sizes can keep higher levels of diversity. This is a good feature in dynamically changing environments as it increases the chances of exploring novel regions of the search space.

VII. DISCUSSION

Enabling robots with different morphologies to acquire a good gait quickly after birth is essential for the evolution of robot morphologies in real time and real space. In this study we examined different versions of the RL PoWER algorithm for this purpose.

Our results showed a difference between employing the RL PoWER algorithm in its original form and using it with modified mutation and crossover operators. The original RL PoWER is fast to learn, but also that it converges very quickly to suboptimal solutions. Reducing the number of parents in the crossover from ten to two improves performance and so does the usage of self-adaptive mutation step-sizes. The overall 'winner' regarding the final solution quality is algorithm D that combines both extensions.

It is interesting to note that for most shapes, smaller bodies tend to move faster. The snake morphology, however does

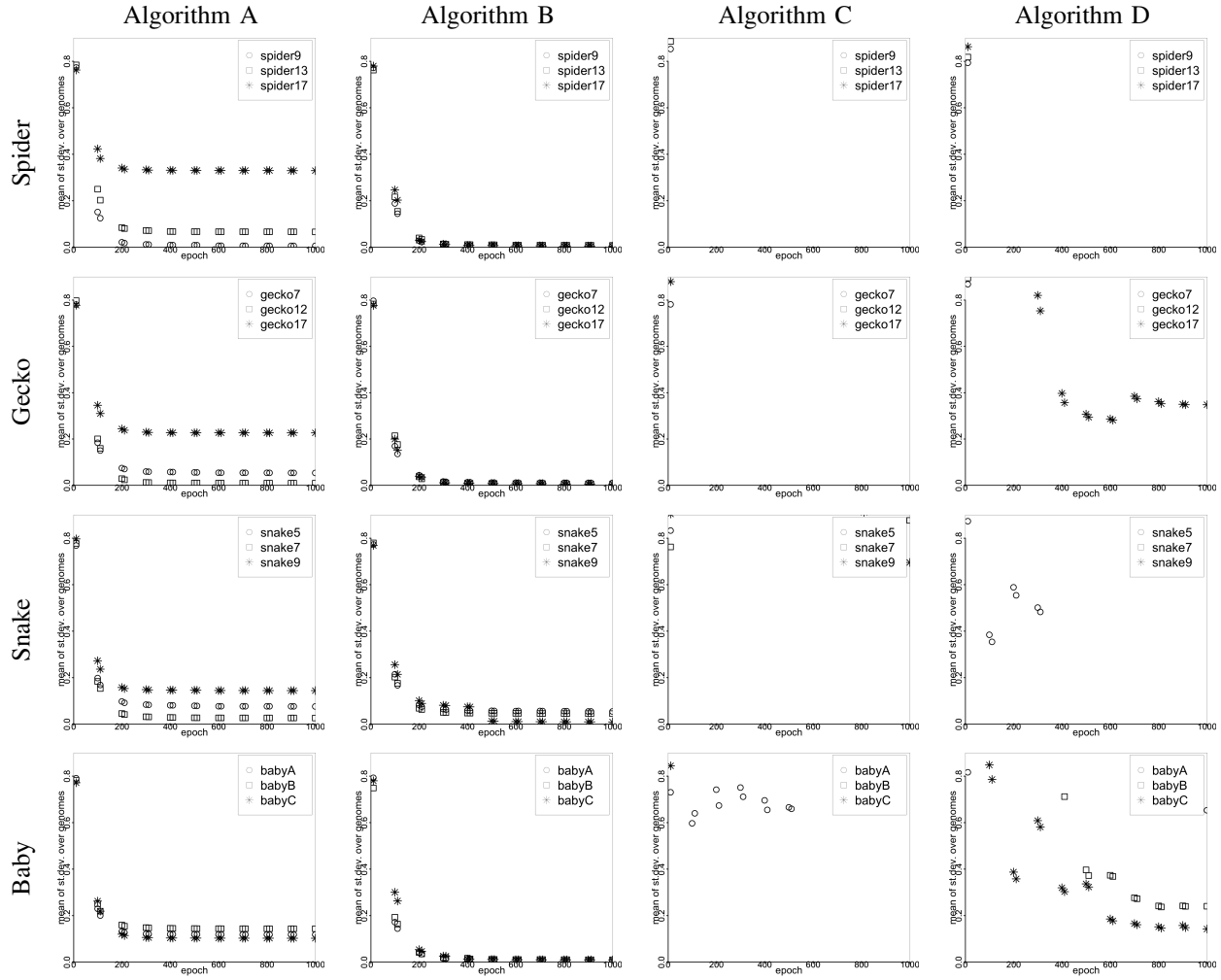

Fig. 4. The distribution of a genetical diversity over time. Each plot shows the mean best fitness for three sizes of a shape (one shape per row, as indicated in the first column) one of the algorithms (one algorithm per column, as indicated in the first row).

move faster with larger shapes. A possible explanation may be that the larger number of actuators in larger bodies increases the amount of interference between moving limbs. For the snake morphology this may be less problematic because all the actuators are aligned (i.e., they move in the same plane). Whether this explanation holds remains to be investigated in future studies.

Future work will be devoted to spline-based controllers capable of handling sensory feedback. With this extension we will be able to handle more complex problems, such as obstacle avoidance and phototaxis.

REFERENCES

[1] A. Eiben, "In Vivo Veritas: towards the Evolution of Things," in *Parallel Problem Solving from Nature – PPSN XIII*, ser. LNCS, T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, Eds., vol. 8672. Springer, 2014, pp. 24–39.

[2] A. Eiben, S. Kernbach, and E. Haasdijk, "Embodied artificial evolution," *Evolutionary Intelligence*, vol. 5, no. 4, pp. 261–272, 2012.

[3] A. Eiben and J. Smith, "From evolutionary computation to the evolution of things," *Nature*, vol. 521, no. 7553, pp. 476–482, May 2015.

[4] A. Eiben, N. Bredeche, M. Hoogendoorn, J. Stradner, J. Timmis, A. Tyrrell, and A. Winfield, "The triangle of life: Evolving robots in real-time and real-space," in *Advances In Artificial Life, ECAL 2013*, P. Liò, O. Miglino, G. Nicosia, S. Nolfi, and M. Pavone, Eds. MIT Press, 2013, pp. 1056–1063.

[5] N. Cheney, J. Bongard, V. Sunspiral, and H. Lipson, "On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures," in *Proc. Artif. Life Conf. 2016 (ALIFE XV)*. Cancun. MX: MIT Press, 2016, pp. 226—-234.

[6] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2112–2118.

[7] B. Weel, M. D'Angelo, E. Haasdijk, and A. Eiben, "On-line gait learning for modular robots with arbitrary shapes and sizes," *Artificial Life Journal*, 2016 (in press).

[8] J. Auerbach, D. Aydin, A. Maesani, P. Kornatowski, T. Cieslewski, G. Heitz, P. Fernando, I. Loshchilov, L. Daler, and D. Floreano, "Robogen: Robot generation through artificial evolution," in *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, no. EPFL-CONF-200995. The MIT Press, 2014, pp. 136–137.

[9] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

[10] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through

continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[11] M. Yim, W. M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 43–52, 2007.

[12] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.

[13] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving coordinated quadruped gaits with the hyperneat generative encoding," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 2764–2771.

[14] E. Haasdijk, A. Rusu, and A. Eiben, "Hyperneat for locomotion control in modular robots," *Evolvable Systems: From Biology to Hardware*, pp. 169–180, 2010.

[15] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. Zagal, and H. Lipson, "Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization," in *Proceedings of the 20th European Conference on Artificial Life*, 2011, pp. 11–18.

[16] D. J. Christensen, U. P. Schultz, and K. Støy, "A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots," *Robotics and Autonomous Systems*, vol. 61, no. 9, pp. 1021–1035, 2013.

[17] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, "Distributed adaptive locomotion by a modular robotic system, m-tran ii," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2370–2377.

[18] M. DAngelo, B. Weel, and A. Eiben, "Online gait learning for modular robots with arbitrary shapes and sizes," in *International Conference on Theory and Practice of Natural Computing*. Springer, 2013, pp. 45–56.

[19] M. D'Angelo, B. Weel, and A. Eiben, "Hyperneat versus rl power for online gait learning in modular robots," in *Proceedings of EvoApplications 2014: Applications of Evolutionary Computation*, ser. Lecture Notes in Computer Science, A. Esparcia-Alcázar, Ed., no. 8602. Springer, Berlin, Heidelberg, New York, 2014, pp. 777–788.

[20] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *Gnu Scientific Library: Reference Manual*. Network Theory Ltd., 2009.

[21] H. Shen, J. Yosinski, P. Kormushev, D. G. Caldwell, and H. Lipson, "Learning fast quadruped robot gaits with the RL PoWER spline parameterization," *Cybernetics and Information Technologies*, vol. 12, no. 3, pp. 66–75, 2012.

[22] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Springer, 2015.

[23] A. E. Eiben, "Multiparent recombination in evolutionary computing," in *Advances in Evolutionary Computing*, ser. Natural Computing Series, A. Ghosh and S. Tsutsui, Eds. Springer, 2002, pp. 175–192.

[24] K. De Jong and J. Sarma, "On decentralizing selection algorithms," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. Eshelman, Ed. San Francisco, CA: Morgan Kaufmann, 1995, pp. 17–23. [Online]. Available: citeseer.ist.psu.edu/dejong95decentralizing.html